Encoder-Decoder Approach to Predict Airport Operational Runway Configuration

A case study for Amsterdam Schiphol airport

Ramon Dalmau & Floris Herrema
Network (NET) and Airport (APT) Research Units
EUROCONTROL Experimental Centre (EEC)
Brétigny-Sur-Orge, France

***Abstract*—The runway configuration of an airport is the combination of runways that are active for arrivals and departures at any time. The runway configuration has a major influence on the capacity of the airport, taxiing times, the occupation of parking stands and taxiways, as well as on the management of traffic in the airspace surrounding the airport. The runway configuration of a given airport may change several times during the day, depending on the weather, air traffic demand and noise abatement rules, among other factors. This paper proposes an encoder-decoder model that is able to predict the future runway configuration sequence of an airport several hours upfront. In contrast to typical rule-based approaches, the proposed model is generic enough to be applied to any airport, since it only requires the past runway configuration history and the forecast traffic demand and weather in the prediction horizon. The performance of the model is assessed for the Amsterdam Schiphol Airport using three years of traffic, weather and runway use data.**

***Index Terms*—runway configuration, machine learning**

I. Introduction

Runways enable aircraft to take-off and landing. As such, at major airports, the capacity of runways is the most restricting element in considering total airport operations. Major airports have several runways in order to accommodate a large amount of aircraft movements. For instance, the Hartsfield-Jackson Atlanta International Airport, which is the busiest airport in the world with more than 107 million passengers in 2018 utilises 5 runways. The number of runways of an airport, however, does not only depend on the volume of traffic. Runways may be reserved for certain type of traffic, wind directions, visibility conditions, time periods, or for noise abatement procedures.

It should be noted that not all runways of an airport are used simultaneously. The planned combination of runways that are active at any time is called the ”runway configuration”. For airports with multiple runways the number of possible configurations may be large. For instance, Amsterdam Schiphol Airport (EHAM) has 6 runways, which could be combined in more than 100 different ways. In practise, however, only 2 or 3 are used simultaneously. During daily time in summer, only 8 runway configurations are used 70% of the time in average[1].

The runway configuration greatly influences the capacity of the airport. In addition, the in and out taxi times and thus the duration of the flights from off-block time to on-block, also depend on the runways that are used to take-off and land from/to the origin and destination airports, respectively. For instance, aircraft using the 18R/36L runway of EHAM (see Fig. 3), located to reduce the noise impact on the surrounding communities, have a 20-minute taxi to/from the terminal, while for other runways, the taxi-time is about 10 minutes. The runway configuration also has a major impact on the occupation of parking stands and taxiways, as well as on the management of traffic in the airspace surrounding the airport.

At present, the selection of runway configuration is mainly based on human experience. Air Traffic Controllers (ATCOs) examine several factors to determine the *best* sequence of runway configurations to be used in the next hours. For instance, it is well known that wind speed and wind direction influence the choice of the runway configuration since cross-winds (relative to the direction of that runway) exceeding a threshold may not be adequate to take-off and landing. Moreover, certain runways may be prohibited under poor visibility conditions due to unqualified instrumentation. In addition to winds and visibility, a runway may be also not operable due to highly intense precipitation or icing conditions. Last but not least, extremely hot temperatures can make take-off impossible for certain aircraft because of insufficient lift force. The scheduled number of arrivals and departures for the next hours as well as the aircraft wake categories are also expected to have a great impact when deciding which runways to use at a given time.

Unfortunately, it is difficult to represent the criteria considered by the ATCOs to determine the runway configuration given the influencing factors with generic rule-based models, and thus it is difficult to accurately predict the future runway configuration (and all the variables that depend on it). Moreover, each airport may implement different rules, meaning that creating a generic rule-based model might be unfeasible. For instance, at EHAM, weather is the main factor considered by ATCOs when selecting the configuration[1]. The environmental (in particular noise) rules for the use of runways, however, also play a role in determining the runway configuration.

Several works have attempted to predict the future runway configuration. For instance, reference [1] proposed a discrete-choice model of the configuration selection process from em-

[1] https://ext.eurocontrol.int/airport_corner_public/EHAM

ISSN 0770-1268

pirical data. Results showed that, if the actual traffic demand and weather conditions where known 3 hours in advance, the model could predict the runway configuration at La Guardia and San Francisco airports with an accuracy of 82% and 85%, respectively. More recently, Ref. [2] proposed an Artificial Neural Network (ANN) architecture that uses similar data to predict the runway configuration and the corresponding capacity. Results for predictions one hour ahead showed a promising predictive power. The dataset used to train and evaluate the model, however, comprised only 24 hours.

This paper proposes an encoder-decoder model inspired by sequence to sequence techniques to predict the runway configuration sequence of an airport every 15 min in an horizon of 6 hours. The model takes as inputs the observed weather and traffic demand in the near past, as well as the weather and traffic demand forecast in the 6-hours prediction horizon. This information is combined with the recently used runway configuration history to perform the sequence prediction. The performance of the model is assessed for EHAM using three years of traffic, weather and runway use data.

II. Sequence to sequence (seq2seq) models

Note that throughout this paper, and as a general rule, scalars and vectors are denoted either with lower or upper case letters, e.g., a or A. Vectors are denoted with the conventional overhead arrow, e.g., $\vec{a}$; while sequences use the same font but in bold, .e.g, $\boldsymbol{a}$. Sets are denoted using calligraphic fonts, e.g., $\mathcal{A}$; while matrices use the same font but in bold, e.g., $\boldsymbol{\mathcal{A}}$.

Seq2seq models lie behind numerous applications such as neural machine translation, text summarisation, speech recognition, video captioning, online chat-bots and other cases where it is desirable to generate a sequence from another [4].

The goal of a seq2seq model is to find the output sequence $\boldsymbol{y} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_{T_y})$ in $\Re^{n_y}$ that maximizes the conditional probability of $\boldsymbol{y}$ given the input sequence $\boldsymbol{x} = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{T_x})$ in $\Re^{n_x}$, i.e., $\text{argmax}_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})$. T_x and T_y are the length of the input and output sequences, respectively.

A. Basic encoder-decoder

Conventional seq2seq models used in many applications consist of two parts: the *encoder* and the *decoder*. The role of the encoder is to condense the information of the input sequence $\boldsymbol{x}$ into a vector of a fixed length $\vec{c}$, commonly known as *context*. The context is used to condition the decoder, which generates the output sequence $\boldsymbol{y}$ that maximises $p(\boldsymbol{y}|\boldsymbol{x})$. The architecture of a basic encoder-decoder is shown in Fig. 1a.

The usual approach is to use a Recurrent Neural Network (RNN) as encoder. Roughly speaking, a RNN has a looping mechanism that allows important information to flow from one time step to the next one. This information is stored in the hidden state of the RNN, $\vec{h}_t \in \Re^{n_h}$. At each time step t the RNN takes the inputs vector and updates its hidden state:

$$\vec{h}_t = \text{Recurrent}_{\phi_h, \psi_h, n_h}\left(\vec{h}_{t-1}, \vec{x}_t\right), \tag{1}$$

where Recurrent is a nonlinear function that depends on the RNN type. In Eq. (1), ϕ_h and ψ_h are the activation and recurrent activation functions, respectively. For instance, for a *vanilla* RNN the update is typically performed as [5]:

$$\vec{h}_t = \tanh\left(\boldsymbol{\mathcal{W}}_{hh}\vec{h}_{t-1} + \boldsymbol{\mathcal{W}}_{hx}\vec{x}_t + \vec{b}_h\right), \tag{2}$$

where $\boldsymbol{\mathcal{W}}_{hh} \in \Re^{n_h \times n_h}$ and $\boldsymbol{\mathcal{W}}_{hx} \in \Re^{n_h \times n_x}$ are trainable matrices and $\vec{b}_h \in \Re^{n_h}$ is the (also trainable) bias vector. Note that at $t = 1$ the hidden state depends on the first inputs vector $\vec{x}_1$ as well as the initial hidden state $\vec{h}_0$. The default approach consists of initialising the hidden state with zeros (i.e., $\vec{h}_0 = \vec{0}$). This strategy often works well for seq2seq.

Vanilla RNNs, frequently have the problem of vanishing gradients which, negatively influences the learning of long sequences. To solve vanishing gradients a popular way is to use LSTM (Long Short Term Memory) [6] or Gated Recurrent Unit (GRU) [7].

Generally speaking, the context vector that conditions the decoder to generate $\boldsymbol{y}$ could be any nonlinear function q of the whole sequence of hidden states generated by the encoder:

$$\vec{c} = q\left(\boldsymbol{h}\right) = q\left(\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_{T_x}\right). \tag{3}$$

The most basic encoder-decoder model, however, simply uses the last hidden state as context vector, i.e, $\vec{c} = \vec{h}_{T_x}$.

The decoder component is another RNN trained to predict the next output $\vec{y}_t$ given the context vector $\vec{c}$ and all the previously predicted outputs $\vec{y}_1, \ldots, \vec{y}_{t-1}$. In other words, the decoder defines a probability over the sequence $\boldsymbol{y}$ by decomposing the joint probability into the ordered conditionals:

$$p\left(\boldsymbol{y}|\boldsymbol{x}\right) = \prod_{t=1}^{T_y} p\left(\vec{y}_t|\vec{y}_{<t}, \vec{c}\right) = \prod_{t=1}^{T_y} p\left(\vec{y}_t|\vec{y}_1, \ldots, \vec{y}_{t-1}, \vec{c}\right), \tag{4}$$

where the context vector is an implicit function of $\boldsymbol{x}$ through Eq. (3). The conditional probability of $\vec{y}_t$ is computed as:

$$\vec{p}_t = p(\vec{y}_t|\vec{y}_{<t}, \vec{c}) = \text{Dense}_{\phi_d, n_y}\left(\vec{s}_t\right), \tag{5}$$

where a fully-connected dense layer of n units and activation function ϕ is defined as the operation that applied to any vector of inputs $\vec{x} \in \Re^m$ generates the output $\vec{y} \in \Re^n$ according to:

$$\vec{y} = \text{Dense}_{\phi,n}(\vec{x}) = \phi\left(\boldsymbol{\mathcal{W}}\vec{x} + \vec{b}\right), \tag{6}$$

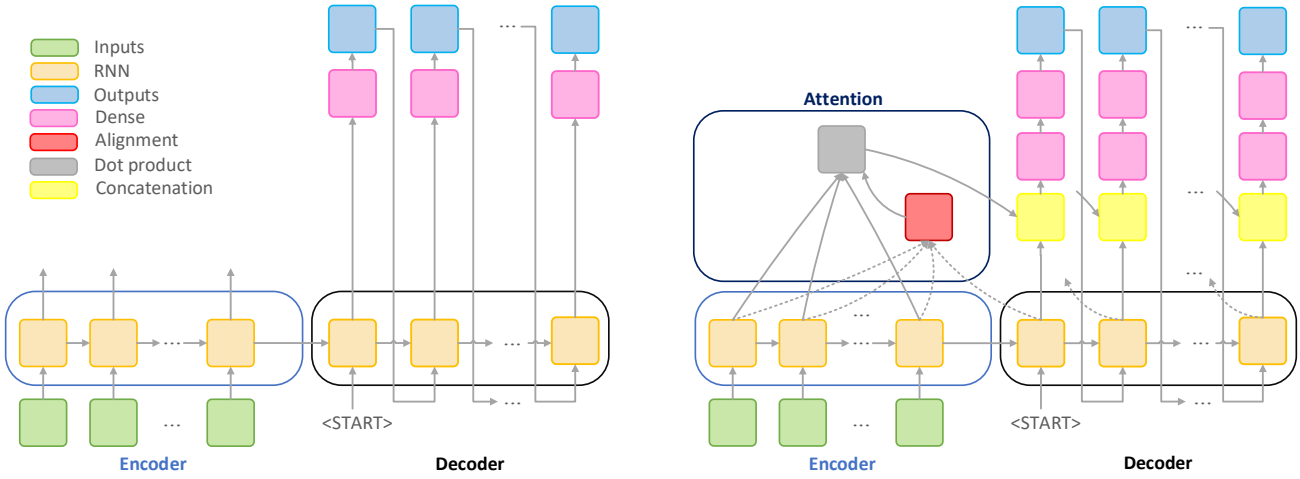
where $\boldsymbol{\mathcal{W}} \in \Re^{n \times m}$ is a weighting matrix, and $\vec{b} \in \Re^n$ is the bias vector. Both $\boldsymbol{\mathcal{W}}$ and $\vec{b}$ are parameters to be trained.

In Eq. (5), $\vec{s}_t \in \Re^{n_s}$ is the hidden state of the decoder's RNN, which can be computed similar to Eq. (1):

$$\vec{s}_t = \text{Recurrent}_{\phi_s, \psi_s, n_s}(\vec{s}_{t-1}, \vec{y}_{t-1}) \tag{7}$$

As for the encoder, in most practical applications the decoder is a GRU or LSTM. In the basic encoder-decoder architecture (see Fig.1a), the context vector is only used once to initialize the hidden state of the decoder (i.e., $\vec{s}_0 = \vec{c} = \vec{h}_{T_x}$).

The decoder behaves differently during training and inference (prediction of unseen data). During training, the ground



(a) Basic (b) With Luong [3] attention

Figure 1. Encoder-decoder architecture (inference case)

truth of the target variable is known and the input at each time step is given as the actual output (and not the predicted output) from the previous time step. During inference (as illustrated in Fig. 1) the input at each time step is the predicted output from the previous time step. According to Eq. (7), at the very first time step of the prediction phase the hidden state is a function of $\vec{y}_0$. Both in training and inference, at the very first time step a *start* token is used as input to initialise the prediction.

B. Encoder-decoder with Luong's global attention

A potential issue with the basic encoder-decoder illustrated in Fig. 1a is that the encoder must compress all the necessary information of the input sequence into a fixed-length vector. When the sequences is very long, the performance of the basic encoder-decoder architecture is critically compromised. In order to solve this issue, advanced encoder-decoder architectures create a different context vector $\vec{c}_t$ for each time step of the decoding phase, instead of building a single context vector out of the last hidden state of the encoder. This architecture is commonly known as encoder-decoder with *attention* [3], [8].

Attention mechanisms are classified into two broad categories: global and local. While global attention uses all the encoder' s hidden states to generate the context vector, local attention uses only a subset of them. For typical Natural Language Processing (NLP) problems, global attention has the drawback of requiring to attend each element on the input sequence for each decoding step, which is computationally expensive and impractical when translating long sequences, e.g., paragraphs or documents. Yet, the length of the input sequences for the model presented in this paper are sufficiently short to be attended by global attention. For this reason, in this paper Luong's global attention [3] has been implemented.

When implementing this attention mechanism, the context vector $\vec{c}_t \in \Re^{n_h}$ at each decoding step is computed as a weighted sum of the hidden states of the encoder:

$$\vec{c}_t = \vec{\alpha}_t \cdot \boldsymbol{h} = \sum_{k=1}^{T_x} \alpha_{tk} \vec{h}_k, \tag{8}$$

where $\vec{\alpha}_t = [\alpha_{t1}, \alpha_{t2}, \ldots, \alpha_{tT_x}]$ is the alignment vector at t, which length equals the number of time steps of the input sequence, i.e., T_x. This alignment vector is computed as:

$$\vec{\alpha}_t = \text{Alignment}(\vec{s}_t, \boldsymbol{h}) = \frac{\exp(\text{score}(\vec{s}_t, \boldsymbol{h}))}{\sum_{k=1}^{T_x} \exp(\text{score}(\vec{s}_t, \vec{h}_k)))}. \tag{9}$$

Several score functions were proposed in Ref. [3]. In this paper, the *general* score function has been implemented because it provided similar results to the *concat* score function yet being simpler and requiring to train less parameters.

At each time step of the decoding phase, the hidden state $\vec{s}_t$ and the corresponding context vector $\vec{c}_t$ are concatenated to produce an attentive hidden state $\vec{a}_t \in \Re^{n_a}$ as follows:

$$\vec{a}_t = \text{Dense}_{\text{tanh},n_a}\left(\vec{c}_t || \vec{s}_t\right), \tag{10}$$

where the operand $||$ refers to vector concatenation. Finally, the attentive hidden state is then passed through a dense layer:

$$\vec{p}_t = p(\vec{y}_t | \vec{y}_{<t}, \vec{c}_t) = \text{Dense}_{\phi_d,n_y}\left(\vec{a}_t\right). \tag{11}$$

The architecture presented so far can be used to solve *multi-class* classification problems, where the output vector at each time step $\vec{y}_t$ has as many elements as possible classes, and the value of each element is the probability of belonging to the corresponding class. For the ground truth, 1 appears at the element of the correct class, and 0 in all the others. Ideally, when a given class is predicted by the decoder, the value of the corresponding element should be close to 1, and the other elements should be close to 0. The most convenient way to accomplish that is to use the softmax activation function as ϕ_d, which assigns a probability to each one of its inputs such

that $\sum \vec{p}_t = 1$. Accordingly, the probability associated to a class is not independent from that of the others.

The encoder-decoder is typically trained to minimize the cross-entropy (CE), calculated independently for each output vector in the sequence and then summed up. For a given training example composed by the input and output sequences:

$$\mathrm{CE}(\boldsymbol{x}, \boldsymbol{y}) = -\sum_{t=1}^{T_y} \vec{y}_t \cdot \log(\vec{p}_t(\boldsymbol{x})), \tag{12}$$

where $\vec{y}_t$ is the ground truth and $\vec{p}_t$ is the probability predicted by the model given $\boldsymbol{x}$. In binary classification, where the number of classes is 2, each element in the output sequence is a scalar, i.e., $\boldsymbol{y} = (y_1, y_2, \ldots, y_{T_y})$ in $\Re$, which probability is predicted by using sigmoid activation function as ϕ_d [5].

III. Proposed Approach

As mentioned in Section I, ATCOs select runway configuration based on many factors, including weather conditions, traffic demand, and noise regulations, which might depend in turn on the hour of the day, for instance. The aim of this study is to assess the feasibility of a data-driven model trained on historical data capable to predict the runway configuration several hours ahead using some of these features as input.

The most straightforward way to accomplish that consists of feeding a feed-forward neural network with the aforementioned features at a particular time and predict the instantaneous runway configuration. Most probably, ATCOs do not only look these features at a given time to make a decision, but also take into account their evolution on a prolonged time interval (e.g., several hours). Therefore, the first hypothesis of the model proposed herein is that a sequence of these features is what drives ATCOs decisions. The second hypothesis is that the probability of selecting a runway configuration at a given time is conditioned on the previous decisions in the recent past. Taking these two hypothesis into account, an attractive model to capture information encoded in a sequence of inputs and generate a sequence out of this is the encoder-decoder with attention described in Section II.

The outputs and inputs of the model are presented in Sections III-A and III-B, respectively. Then, the proposed encoder-decoder architecture is described in Section III-C.

A. Outputs of the decoder

The target variable for the problem tackled in this paper can be defined in different ways. The final choice will determine the number and type of outputs that the decoder generates, as well as the activation function of their last dense layer.

The most straightforward strategy consists of identifying all possible combinations of runways, associate each one to a given class, and solve a *single-output multi-class* classification problem. Using this approach, the output of the decoder at each time sample is, in point of fact, a class representing the predicted runway configuration. As discussed in Section II-B, the usual way to address multi-class classification problems is to transform the target variable to a one-hot vector, meaning that the output of the decoder is a vector with as many elements as classes (runway configurations), where each element is the probability associated to that class. These probabilities are generated by a softmax function as ϕ_d and sum up to 1.

In this paper, the problem is addressed as a *multi-output mutli-class* classification, where each output corresponds to a runway. The airport may have runways that are used only for take-off, only landing, or that could be used for both. For the set of runways that are always used for an unique type of operation ($\mathcal{R}_b$), the corresponding outputs are scalar binary variables: active or inactive. For the set of runways that could be used to accommodate departures and arrivals ($\mathcal{R}_m$), the corresponding outputs are categorical variables with three possible classes: take-off, landing or inactive. The way of handling each individual output for runways in $\mathcal{R}_m$ is identical to that of the single-output multi-class described above. The output corresponding to each runway in $\mathcal{R}_b$ is a scalar which probability value is generated by a sigmoid activation function.

If compared to the single-output multi-class strategy, the multi-output multi-class has the advantage that if only the state of one runway (one output) is unsatisfactorily predicted, the penalty to the loss function is lower than if the predictions for all runways (i.e., runway configuration) are wrong.

The decoder generates the sequence of outputs, where each output is the state of the corresponding runway (either a binary scalar or a one-hot representation of a 3-class categorical variable), for the following 6 hours in intervals of 15 minutes. Therefore, in this model $T_y = 24$, and $n_y = |\mathcal{R}_b| + 3|\mathcal{R}_m|$.

B. Inputs of the encoders

The encoder-decoder model is composed by two encoders, each one receiving a different sequence of input features.

The task of the first encoder is to capture factors influencing the choice of runway configuration sequence related to the weather and traffic demand in the near past and also in the prediction horizon. The length of the sequence fed to this encoder is $T_x = 48$, with each element in the sequence including the features shown in Table I over a 15-minute interval. Accordingly, this encoder receives information over a 12-hour period. The first 24 elements correspond to the observations in the past 6 hours, and the remaining 24 elements include the forecast over the prediction horizon (next 6 hours).

The input vector fed to this encoder at each time step includes weather features such as wind direction and speed, temperature and visibility; demand features such as how many departures of each aircraft category type are planned in the corresponding 15-minute interval; and calendar features such as the hour of the day. Note that some of these features are continuous while others are categorical (discrete). In this study, each categorical variable has been represented as a one-hot vector, yet the use of embeddings is also encouraged.

The introduction of the second encoder is motivated by the fact that the recently used configuration sequence might also condition the decision of ATCOs. As such, the second decoder takes the known runway configuration sequence used in the past 6 hours, also discretised in intervals of 15 minutes.

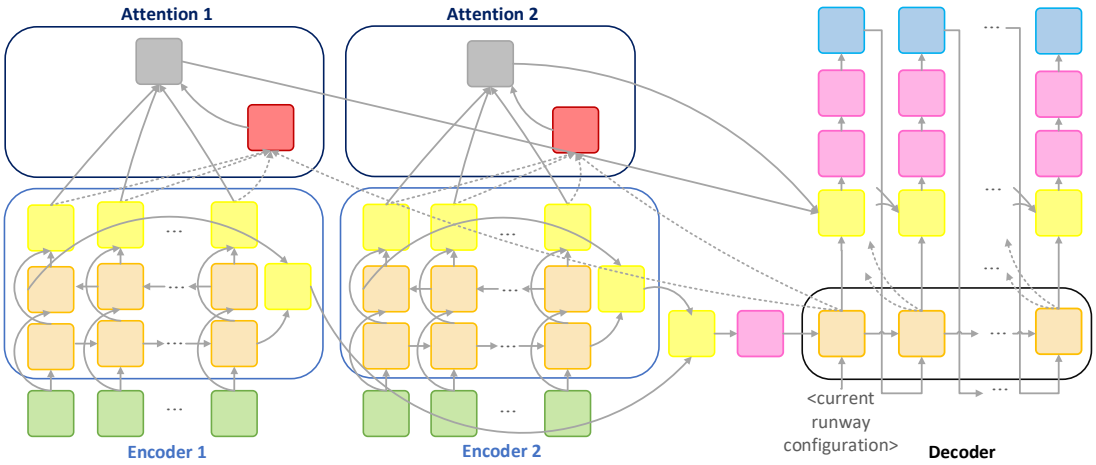


Figure 2. Architecture of the encoder-decoder (same color convention as Fig. 1 applies)

Accordingly, the length of the input sequence fed to the second encoder is $T_x = 24$. Table II lists the features included in the input vector of the second encoder at each time step.

TABLE I. INPUTS TO THE 1ST ENCODER EVERY 15 MINUTES. AIRCRAFT CATEGORIES ARE HEAVY: H, MEDIUM: M, AND LIGHT: L.

Group	Type	Feature
Weather	Continuous	wind direction wind speed wind gust temperature dew point ceiling visibility cloud height
	Categorical	cloud cover cloud type precipitation obscuration
Demand	Continuous	# of H departures in a 15/30/60 min interval # of M departures in a 15/30/60 min interval # of L departures in a 15/30/60 min interval # of H arrivals in a 15/30/60 min interval # of M arrivals in a 15/30/60 min interval # of L arrivals in a 15/30/60 min interval
Calendar	Categorical	Hour of the day Day of the week Week of the month Month of the year

TABLE II. INPUTS TO THE 2ND ENCODER EVERY 15 MINUTES

Group	Type	Feature
Runway use	Categorical (binary)	runway r state $\forall r \in \mathcal{R}_b$
	Categorical (multi-class)	runway r state $\forall r \in \mathcal{R}_m$

C. Architecture of the encoder-decoder model

The architecture of the proposed encoder-decoder is illustrated in Fig. 2. Both encoders are bi-directional RNNs (BRRNs), which simply consists on putting two independent RNNs together. The input sequence is fed in normal time order for one RNN, and in reverse time order for the other. Then, the outputs of the two RNNs are concatenated at each time step, producing a vector in $\Re^{2n_h}$ that receives information from both past and future. The last hidden state of the forward and backward RNNs of both encoders are also concatenated, and passed through a dense layer which number of units must be identical to the number of units in the decoder's RNN (n_s). The output of this dense layer is used as initial hidden state of the decoder. At the first time step of the prediction, the decoder is fed with the current runway configuration and the hidden state generated by the two encoders. Therefore, the token that initialises the prediction is the current runway configuration.

The updated hidden state is fed to two independent attention layers. Each attention layer computes a context vector that hopefully captures the most important information from the corresponding encoder. The two context vectors are concatenated with the hidden state of the decoder and passed through a dense layer to generate the attentive hidden state (see Eq. (10)).

Remember that, in this model, the decoder generates as many outputs as runways, where each output is either binary or a 3-class categorical variable. For each output (runway), the probabilities of the possible runway states are given by a sigmoid or softmax activation function as ϕ_d, respectively.

The weights of the encoder-decoder model could be found by minimising the sum of cross-entropy for each output:

$$L(\boldsymbol{x}, \boldsymbol{y}) = \sum_{r \in \mathcal{R}_b \cup \mathcal{R}_m} \text{CE}(\boldsymbol{x}, \boldsymbol{y}_r) \tag{13}$$

where $\boldsymbol{y}_r$ is the sequence of outputs for the runway r and CE is the cross-entropy (see Eq. (12)).

This works well as long as the dataset is balanced. Dealing with classification problems where the classes are not represented equally is a dangerous situation that could lead to the well known *accuracy paradox*, in which the accuracy of the model is supposed to be excellent, but in reality is only reflecting the underlying class distribution. There are several strategies to deal with unbalanced datasets.

For instance, one could sample more minor class samples or remove major class samples; or try to generate artificial samples for the minor classes using Synthetic Minority Oversampling Technique (SMOTE). In this paper the solution proposed in Ref. [9] has been implemented, which consists of reshaping the loss function to down-weight *easy* examples (i.e., those predicted with a very high probability) and focus on difficult ones. The result is the Focal Loss (FL);

$$\mathrm{FL}(\boldsymbol{x}, \boldsymbol{y}) = -\sum_{t=1}^{T_y} (1 - \vec{p}_t(\boldsymbol{x}))^{\gamma} \cdot \vec{y}_t \cdot \log(\vec{p}_t(\boldsymbol{x})) \qquad (14)$$

where γ is a fixed parameter that smoothly adjusts the rate at which easy examples are down-weighted.

IV. Set up of the experiment

The performance of the model proposed in Section III-C has been assessed for a realistic case study at EHAM using three years of historical data. The dataset is described in Section IV-A. Section IV-B shows the scenario of the experiment.

A. Data set

The data used for the assessment concerns the time period from 1st of January 2016 to 1st of May 2019.

Air traffic demand data (number of arrivals and departures) have been obtained from the Enhanced Tactical Flow Management System (ETFMS), which monitors flight evolution data received from the Network Manager and provides real-time information to all operational stakeholders. The ETFMS data provides the up-to-date status of the flight during its whole life by including, for instance, the Expected Take-Off Time (ETOT), the Actual Take-Off Time (ATOT), the Estimated Time of Arrival (ETA) and the Actual Time of Arrival (ATA).

The aircraft type designators (heavy, medium or light) of each flight scheduled to depart or arrive at the airport has been obtained from the ICAO DOC 8643 specification.

Weather data have been extracted from Meteorological Terminal Aviation Routine Weather Reports (METARs). METARs are typically generated once an hour at the airports or at weather observation stations. A standard METAR includes information about the temperature, pressure, dew point, wind direction and speed, precipitation, cloud cover and height, as well as visibility and ceiling. A METAR may also include information about the presence of specific weather phenomena such as precipitation and obscuration type and intensity.

Finally, the EHAM's runway configuration history has been kindly provided by LVNL (Luchtverkeersleiding Nederland).

These data have been merged into a single dataset according to time. Then, the dataset has been re-sampled in intervals of 15 minutes. The re-sampled dataset has been used to generate sequences every hour, where each sequence includes data for the past and next 6 hours in 15-minute intervals (48 samples).

It is important to remark that, similar to [1], in this experiment actual traffic demand and weather data (i.e., post-ops) have been used as input to the first encoder for the whole 12-hour input sequence, instead of using the forecast. Therefore, the results shown in this paper correspond to a best-case scenario in which the demand and weather forecast in the 6-hours prediction horizon is known with high accuracy.

B. Scenario

Figure 3 illustrates the EHAM layout. During periods of high demand, three runways are typically used simultaneously (e.g., 18R for landing and 24/18L for take-off). In some specific situations, ATCOs may decide to increase the number of active runways to four. When the traffic demand is low, two runways are typically in use (e.g., 18R for landing and 24 for take-off). In exceptional cases, only one runway may be active, which is then used for both departures and arrivals.

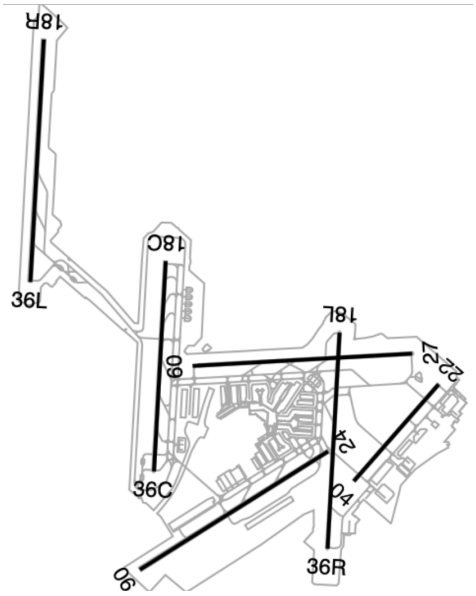


Figure 3. Amsterdam Schiphol Airport (EHAM) layout

The number of different runway configurations observed for the time period considered in this experiment was 142, from which only 20 of them where used more than 1% of the time. The sum of frequencies for these 20 runway configurations was 87%. Given the limited amount of data, only time periods in which one of these 20 runway configurations was active were included in the dataset, and the remaining were excluded.

Table III shows the state frequency of each runway, after removing the 13% of time periods in which *rare* runway configurations were used. Note that each row sums up to 1.

V. Results

Typical strategies to assess the performance of the model, such as using randomised train-test splits and k-fold cross-validation, are not appropriate for time series prediction because the model may be trained on data from future to predict past. The prediction of the past knowing the future, like other types of data leakage, overestimates the quality of the model. Walk-forward evaluation is a well-known method to correctly evaluate a model for time series prediction by respecting the temporal order of the train-validation-test split [10].

TABLE III. NORMALISED RUNWAY STATE FREQUENCY

RWY	Frequency			type
	Inactive	Landing	Takeoff	
06	0.721	0.279	-	binary
09	0.982	-	0.018	
18L	0.755	-	0.245	
18R	0.445	0.555	-	
24	0.467	-	0.533	
27	0.897	0.103	-	
36L	0.609	-	0.391	
36R	0.877	0.123	-	
36C	0.848	0.066	0.086	multi-class
18C	0.807	0.164	0.029	

Initially, the model was trained with data from 1st January 2016 to 1st January 2019, but using the last 10% of the training data for validation. After training, the model was used to predict the runway configuration sequences for the 2nd January 2019. These predictions along with their corresponding ground truth were stored for performance evaluation. Then, data for the 2nd January 2019 was appended to the train set, the validation set was updated to include the last 10% of the train data, and the model was re-trained. This process was repeated day after day until 1st May 2019. The model was evaluated using the predictions from 2nd January 2019 to 1st May 2019.

Section V-A shows the hyperparameters of the model, which were selected using the initial train-validation set. Section V-B shows an example that illustrates a practical application of the model. Finally, Section V-C shows the performance metrics of the model as a result of the walk-forward evaluation.

A. Optimal hyperparameters

Given the relatively small amount of hyperparameters of the model, the ones that better performed in the initial validation set were selected using manually fine-tuning (see Table IV). However, other techniques such as grid search, randomised search or Tree Parzen Estimators (TPE) are also recommended.

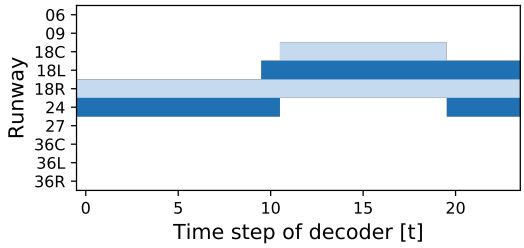
TABLE IV. HYPERPARAMETERS OF THE MODEL

hyperparameter	value
Attention score function	general (see Ref. [3])
Type of RNN encoder / decoder	LSTM / LSTM
Units of the encoder's RNN (n_h)	16
Units of the decoder's RNN (n_s)	32
Attention vector length (n_a)	32
batch size	64
learning rate	0.001
γ (see Eq. 14)	2
Training epochs / early stopping patience	50 / 5

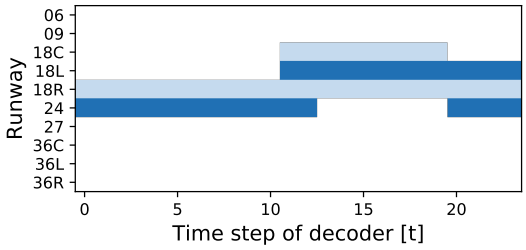
B. Illustrative example

Figures 4a and 4b show a 6-hours runway configuration prediction and the corresponding ground truth, respectively.

According to Fig. 4, the proposed encoder-decoder model was able to accurately predict that runway 18R would be used for landing during the next 6 hours. The model also identified



(a) Prediction



(b) Ground truth

Figure 4. Example (light blue: Landing, dark blue: Take-off)

that runway 18C would be also active during a limited time period, probably to accommodate a higher demand of arrivals. Similarly, the model was able to accurately estimate the switching times for the runways used for take-off. Fig. 4 shows that, even if not being perfect, the state prediction for each output (here a runway) closely followed the actual sequence.

Figure 5 shows the alignment matrix ($\boldsymbol{\mathcal{A}} \in \Re^{T_y \times T_x}$) of the first encoder for this illustrative example. Each row of this matrix corresponds to the alignment vector $\vec{\alpha}_{tq}$ of a time step t of the decoding phase. Roughly speaking, the first 24 columns of this matrix are the weights associated to the observed weather and traffic demand (condensed into the hidden state of the encoder) in the past 6 hours, and the last 24 columns those associated to the forecast weather and traffic demand in the 6-hours prediction horizon. The higher the weight, the more attention the decoder put on the corresponding hidden state of the first decoder to perform the prediction. According to Fig. 5, and as expected, the decoder gives more importance to the weather and traffic demand in the prediction horizon (last 24 columns). Furthermore, more attention is given to the input features when close to time step for which the runway state has to be predicted. In other words, weights are clustered along the diagonal of the right 24×24 square of Fig. 5.

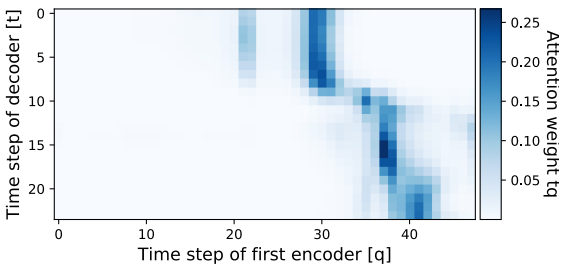


Figure 5. Attention weights

C. Aggregated performance metrics

These performance of the model on unseen data have been evaluated by comparing the predictions performed during the walk-forward validation with their respective ground truth. Table V shows the classical metrics used to assess the performance of classification models: precision, recall and f1-score. A detailed description of these metrics can be found in [5].

TABLE V. PERFORMANCE METRICS FOR DIFFERENT PREDICTION LOOK-AHEAD TIME INTERVALS (IN HOURS)

RWY	precision			recall			f1-score		
	(0,2]	(2,4]	(4,6]	(0,2]	(2,4]	(4,6]	(0,2]	(2,4]	(4,6]
06	0.98	0.96	0.96	0.98	0.97	0.97	0.98	0.97	0.96
09	0.99	0.99	0.99	1.0	1.0	1.0	0.99	1.0	0.99
18C	0.9	0.86	0.83	0.9	0.86	0.84	0.9	0.86	0.84
18L	0.86	0.82	0.82	0.91	0.9	0.88	0.89	0.86	0.85
18R	0.92	0.87	0.84	0.91	0.85	0.8	0.91	0.86	0.82
24	0.9	0.88	0.88	0.83	0.73	0.71	0.86	0.8	0.79
27	0.98	0.97	0.96	0.98	0.97	0.97	0.98	0.97	0.96
36C	0.95	0.94	0.94	0.96	0.94	0.94	0.96	0.94	0.94
36L	0.97	0.95	0.95	0.97	0.96	0.96	0.97	0.96	0.95
36R	0.99	0.98	0.98	0.98	0.97	0.97	0.98	0.98	0.97

According to Table V, the precision of the model is higher than 0.86 for all runways when performing predictions up to 2 hours ahead. It should be noted that this metric basically answers the question to: "Of all runway states that the model predicted as active, how many actually happened?". Interestingly, Table V also shows that the precision of the model does not significantly degrade even if extending the prediction horizon up to 6 hours. This surprisingly stable precision score may be due to the assumption that a perfect weather and traffic demand forecast was available in the whole prediction horizon.

Similar conclusions apply to the recall, which basically answers the question to: "Of all the runways that truly were active, how many did the model identify?". For all the runways, the model is able to predict the state in the next 2 hours with a high recall. The worst case is observed for the runway 24 which, as expected, corresponds to the output with more balanced runway states (47% of the time was inactive and 53% used for take-off). The remaining runways show excellent recall figures above 0.91. The recall also decreases as the prediction look-ahead time increases. Yet, and as observed for the precision score, its degradation is not significant.

Finally, f1-score is the weighted average (i.e., harmonic mean) of precision and recall such that the lowest value is highlighted. In other words, if the precision or the recall is small, the other metric no longer matters. Using f1-score as a metric, one can ensure that if its value is high, both precision and recall of the model reveal model's quality. According to Table V, the f1-score is higher than 0.86 for all runways when predicting their state 2 hours ahead. As for the recall, the worst score correspond to the runway with the most balanced states (runway 24). This metric also showed to be relatively constant regardless of the look-ahead time of the prediction.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed an encoder-decoder model that is able to accurately predict the runway configuration sequence of an airport 6 hours ahead using only historical data.The model was assessed for Amsterdam Schiphol airport using three years of historical weather, traffic demand and runway use data, showing excellent figures of classification performance in the whole prediction horizon. This model could provide more accurate capacity and taxi-times figures to all operational stakeholders. Moreover, it could feed other models to predict, for instance, the Standard Instrumental Departure (SID).

However, predictions were performed assuming accurate forecast of weather and traffic demand in the prediction horizon. Future work will evaluate the performance degradation due to the use of realistic weather and traffic demand forecast. It is also foreseen to investigate the performance of simplified versions of the model, such as including only features in the prediction horizon (i.e., remove the information about past runway configurations, traffic demand and weather), and/or to completely remove the second encoder from the model.

ACKNOWLEDGMENT

This work is supported by EUROCONTROL and the aviation industry. The authors would like to thank LVNL for providing access to the runway configuration data. The authors also acknowledge the heads of Network and Airport Research Units at EUROCONTROL, Franck Ballerini and Bob Graham, respectively, for their valuable insights.

REFERENCES

[1] J. Avery and H. Balakrishnan, "Predicting Airport Runway Configuration: A Discrete-Choice Modeling Approach," in *11th USA/Europe Air Traffic Management Research and Development Seminar (ATMS2015)*, Lisbon, Portugal, June 2015.

[2] M. S. Ahmed, S. Alam, and M. Barlow, "A Multi-Layer Artificial Neural Network Approach for Runway Configuration Prediction," in *Proceedings of the 8th International Conference on Research in Air Transportation (ICRAT 2018)*, Castelldefels, Spain, June 2018.

[3] T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, September 2015, pp. 1412–1421.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, Montreal, CA, 2014.

[5] A. Geron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017.

[6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *EMNLP*. ACL, 2014, pp. 1724–1734.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, May 2015.

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[10] D. Falessi, L. Narayana, J. F. Thai, and B. Turhan, "Preserving Order of Data When Validating Defect Prediction Models," 2018.